Hindawi
Shock and Vibration
Volume 2021, Article ID 9937846, 15 pages
https://doi.org/10.1155/2021/9937846

Research Article

Remaining Useful Life Estimation through Deep Learning Partial Differential Equation Models: A Framework for Degradation Dynamics Interpretation Using Latent Variables

Sergio Cofre-Martel[1] **Enrique Lopez Droguett**[1,2] **and Mohammad Modarres**[1]

[1]*Department of Mechanical Engineering, Center of Risk and Reliability, University of Maryland, College Park, MD, USA*
[2]*Department of Civil and Environmental Engineering & The Garrick Institute for the Risk Sciences, University of California, Los Angeles, USA*

Correspondence should be addressed to Sergio Cofre-Martel; scofre@umd.edu

Received 19 March 2021; Accepted 19 May 2021; Published 27 May 2021

Academic Editor: Tangbin Xia

Remaining useful life (RUL) estimation is one of the main objectives of prognostics and health management (PHM) frameworks. For the past decade, researchers have explored the application of deep learning (DL) regression algorithms to predict the system's health state behavior based on sensor readings from the monitoring system. Although the state-of-art results have been achieved in benchmark problems, most DL-PHM algorithms are treated as black-box functions, giving little-to-no control over data interpretation. This becomes an issue when the models unknowingly break the governing laws of physics when no constraints are imposed. The latest research efforts have focused on applying complex DL models to achieve low prediction errors rather than studying how they interpret the data's behavior and the system itself. This paper proposes an open-box approach using a deep neural network framework to explore the physics of a complex system's degradation through partial differential equations (PDEs). This proposed framework is an attempt to bridge the gap between statistic-based PHM and physics-based PHM. The framework has three stages, and it aims to discover the health state of the system through a latent variable while still providing a RUL estimation. Results show that the latent variable can capture the failure modes of the system. A latent space representation can also be used as a health state estimator through a random forest classifier with up to a 90% performance on new unseen data.

1. Introduction

As the evolution of traditional condition-based maintenance (CBM) techniques, prognostics and health management (PHM) frameworks seek to study and predict the evolution of a system's health state based on data collected from sensor readings. This data is expected to contain critical information related to the system's past and current health state [1]. The main goal of a PHM framework is to estimate the remaining useful life (RUL) of the system, which is later used as a metric for decision-making during the optimization of maintenance policies and health management [1, 2]. Obtaining accurate RUL estimations from sensor data requires a precise knowledge and understanding of the system and, depending on the available information, three main approaches can be implemented for the RUL estimation: physics-based models (PBMs) [3], data-driven approaches (DDAs) [4], and hybrid methods [5]. In this context, we present a deep learning framework to uncover the physics of complex systems' degradation. The framework is inspired by physics-informed neural networks and can be considered a hybrid method for the health state assessment and RUL estimation.

Hybrid methods combine PBMs and DDAs to overcome their weaknesses and combine their strengths [5, 6]. On the one hand, PBMs rely on a mathematical representation to describe the degradation physics governing the system. These methods require a few data points for the training process and yield results directly interpretable by the user. Although PBMs are highly accurate and reliable, they are system-dependent models and cannot be easily scaled and adapted from one system to another. This is why PBMs reliability prognostics studies are usually limited to local crack propagation and corrosion [3], making their direct application to complex systems a challenging task.

On the other hand, machine learning [7] and deep learning (DL) [8] have become the preferred application of DDAs to PHM. These techniques provide an alternative to analyze complex systems when the physics behind the degradation process is unknown. These can extract abstract information and features from massive datasets while building and discovering complex functional and temporal relationships from the data [9]. Deep learning approaches have been implemented in a great variety of systems for prognostics purposes, such as lithium-ion batteries state of health (SOH) and state of charge (SOC) estimation, [10–13], RUL estimation in rolling bearings [14–16], and turbofan engines [17–20].

Although great advances have been made in DL applications to PHM, there are still many challenges to face before implementing these models in the industry [2, 9]. One of these challenges is model interpretability, as DL applications create explainable models that cannot be directly interpreted by the end-user. This has had a detrimental effect on the engineers' trust to implement DL models in real-life systems [21]. Without interpretability, one can only rely on performance metrics to select a model. This can bias the user to choose models with a low error on their training and validation data, regardless of the model's true representation of the system under study. In this regard, third-party software and packages have been developed, providing information on feature relevance for models' predictions [22, 23]. For instance, in [21], the authors presented an algorithm called Local Interpretable Model-Agnostic Explanations (LIME) that provides insight into the relevance that input features have on an ML classification model's prediction. A similar framework was presented by Lundberg and Lee [22] called Shapley Additive explanations (SHAP) for deep learning models. This framework assigns weight values to the input features as importance measures of their effect on the DL model's output. These third-party algorithms provide valuable information for the models' interpretability: nevertheless, they primarily address classification models focusing on natural language or image processing and cannot be implemented within the model itself. Such algorithms can be used as preprocessing or postprocessing techniques. However, they do not influence the model's performance as feature relevance does not have any influence on the models' learning process.

In the context of DL-PHM models, two elements heighten the barriers for model interpretability that are yet to be addressed: the use of time as an explicit variable and the explicit relationship between the physics of the system and the input variables of the model. Indeed, most of the DL-PHM models do not explicitly consider time as a variable in their calculations. Works that apply recurrent neural networks (RNN) and its long-short term memory (LSTM) variation [24–26] use input data with time implicitly embedded through consecutive feature logs, which are then interpreted by the model. Here, the network is trained with a sequence of data points to understand the time scale represented in the data. Thus, the network is given the additional task of interpreting the time relationship among its features. However, new unseen data logs might have different temporal behavior in their log sequences. Likewise, embedding the physics of degradation of a system to a DL framework is a challenging task. Although advances have been made in this area [27, 28], solutions heavily rely on the availability of an empirically based mathematical model (i.e., crack propagation and corrosion, resp.) to describe the damage propagation or future behavior of the system degradation.

The latest advances in DL algorithms have shown that it is possible to embed partial differential equations (PDEs) to DL models. Raissi et al. [29] presented a physics-informed neural network (PINN) framework to solve PDEs by incorporating them as a penalization term to the cost function during the neural network (NN) training process. The framework also allows us to discover PDEs embedded in the data when an explicit equation is not available. This opens the door to create a dynamic relationship between the sensor data and the degradation process in complex systems using DL models in PHM. In this paper, we present a deep neural network (DNN) framework for RUL prognostics that maps the monitoring data and time to a latent variable representation linked to the system's degradation dynamics through a PDE-like penalization function. Once the model is trained, the latent space representation works as a system health estimator quantitatively and qualitatively. In other words, this framework resembles a PDE, where, given initial feature values (i.e., initial conditions), the algorithm can estimate a RUL value through the PDE solution for a given time after the given initial conditions.

Up to date, most DL applications to PHM focus on either diagnostics or prognostics. Very few research works have provided frameworks that can perform these two tasks simultaneously. For instance, Kim and Sohn [30] presented a multitask deep CNN with double outputs, one for prognostics and another for diagnostics. This requires manually handcrafting labels and significantly increases the number of trainable parameters. The training of RNN models requires input data shaped as time windows, which can be impractical to create when sensor data is not sampled at a constant rate or contains missing data points, which is common in real case scenarios. Time windows can also be a source of overfitting if the preprocessing of the data is not carefully done. Further, none of the aforementioned frameworks provide interpretation or visualization of their results. As such, the contributions of this paper are the following:

(1) We present a framework that aims to bridge the gap between statistics-based and physics-based PHM applications.

(2) Inspired in PINN, the proposed framework uses a dynamic PDE-like penalization function that explicitly binds the monitoring data and time to the system's degradation process. This is the first application of PINN to DL-PHM frameworks to the authors' best knowledge.

(3) By using time explicitly, the framework is able to capture the temporal behavior of the data directly. This differs from other commonly used DL algorithms in PHM frameworks such as convolutional neural networks (CNNs) and long short-term memory (LSTM) neural networks which infer these relationships from the data structure instead.

(4) The framework delivers a combined diagnostics and prognostics analysis of a system by providing a RUL estimation along with a health classifier between the system's healthy and degraded states.

(5) The proposed framework also provides interpretability of the system's health state through the visual representation of a latent variable.

The remainder of this paper is structured as follows: Section 2 presents the background behind PDEs applied to DL. Section 3 discusses the proposed DL framework, which is trained with the dataset presented in Section 4. The obtained results and their discussion are presented in Section 5. Section 6 outlines the main conclusions and remarks of this study.

2. Physics-Informed Deep Learning

Most DL algorithms' applications are implemented as black-box functions in which the extraction of abstract relationships in the data is left for the machine to find. In this regard, efforts have been made to provide both interpretation and constraints to these techniques from a physics perspective. Raissi et al. [29] proposed a physics-informed neural network framework that integrates and solves PDEs given a set of initial and boundary conditions. In this work, the authors show that a PINN framework can also be used to recover or create PDEs from the data itself without any prior underlying knowledge on the physics governing the system under study. To understand how this algorithm works, it is necessary to quickly review the architecture behind DL models as function representations and the principles of PDEs.

The main structure in DL is deep neural networks. Here, an input value is evaluated through sequential combinations of nonlinear functions to yield the desired output value. Hence, one can represent the output y of a NN as a function in the form of

$$\hat{y} = f(X, W), \tag{1}$$

where $f(X, W)$ is the NN, X are the input values, and W is a tensor of parameters called weights, which defines the function. Two key components compose a NN: layers and hidden units (also known as neurons). A layer is a nonlinear function of an input value, commonly represented as

$$h_i = \sigma\left(W_i^T h_{i-1} + b_i\right), \tag{2}$$

where h_i is the hidden layer i, represented by its weight matrix W_i and bias vector b_i. Notice that the relationship among h_i, W_i, and X is a simple linear regression. This is then evaluated in a nonlinear function σ, also referred to as activation function. The dimensions of the weight matrix for a NN layer are determined by the number of neurons from the previous layer and its number of neurons. As it can be observed in equation (2), a layer takes as input the output of the previous layer, and it yields an output, which then goes on into the next hidden layer, and so on until the output layer is reached. For instance, equation (3) shows a two-layer NN of input X, output $\hat{y}$, and activation function σ:

$$\hat{y} = \sigma\left(W_2^T \sigma\left(W_1^T X + b_1\right) + b_2\right) = f(X, W). \tag{3}$$

Thus, for a given dataset (X, y), the parameters defining the NN in equation (3) are optimized to minimize the average of the squared errors, which is the so-called loss function described in equation (4). Given a set of data points (often referred to as dataset), equation (4) can be optimized using gradient descent [31] and backpropagation [32]:

$$\text{loss} = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - y_i\right)^2. \tag{4}$$

On the other hand, PDEs model the behavior of a function of interest based on the relationship between its partial derivatives with respect to its input variables. For instance, let $u(z, t)$ be a two-dimensional function of space and time. Then, a PDE for $u(z, t)$ can be represented as

$$u_t = F(z, u, u_z, \ldots), \tag{5}$$

where subindexes indicate partial derivatives of the function $u(z, t)$, for example, $u_z = \partial u(z, t)/\partial z$. The right-hand side of equation (5) is represented by a function F with input variables related to the space variable. In their proposed methodology for PINNs, Raissi et al. took advantage of automatic differentiation [33] to formulate a PDE-like penalization function by considering the target variable of interest by $u(z, t)$ (e.g., velocity field, temperature) as the output of a NN that takes z and t as input variables. As such, one can use automatic differentiation to calculate the exact derivative of the NN representing u with respect to any of its input variables (e.g., u_z, u_t). This allows the creation of a PDE in the form of equation (5), where u_t is the time derivative of the output variable. The function F on the right-hand side is represented by a second NN, which takes the spatial variables and their corresponding u derivatives as input variables. Equation (5) can be written as a cost function in terms of a function f described in equation (6). Adding f as a penalization term to the training cost function of the DL model would then bind the behavior of the parameters representing the NNs of $u(z, t)$ and F through the PDE. Note that if the right-hand side function $F(z, u, u_z, \ldots)$ in equation (5) is known, we can directly implement it to the training cost function. Hence, the optimization cost function of the neural network can be written as shown in equation (7), as the sum of the loss function in equation (4), and the square of f in equation (6). Here, λ is the weight value for the penalization function, and M is the number of points to be tested in the PDE. These can be collocation points, initial conditions, or boundary conditions:

$$f u_t - NN(z, u, u_z, \ldots) = 0. \tag{6}$$

$$\text{Cost} = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - y_i\right)^2 + \lambda \frac{1}{M} \sum_{j=1}^{M} f^2. \tag{7}$$

One of the first implementations of NNs to approximate PDEs was presented in [34], focused on the numerical challenges presented by nonlinear PDEs on continuous mechanical systems. Here, the output of a DNN was used as an approximation for the solution of the PDE (i.e., u), and an unconstrained optimization function was enforced at specific layers and neurons of the network. This architecture is used to solve a linear Poisson equation and thermal conduction with a nonlinear heat generation problem. Later research showed applications of DNNs to solve general coupled PDEs based on Dirichlet and Newman boundary conditions [35]. These first studies mostly focused on the computational efficiency of using NNs to solve PDEs when compared to traditional methods such as finite-element analysis. However, at the time, studies were limited by computational hardware capabilities. Given the nature of their definition, PINNs have mostly been applied in the fluid dynamic research community. The case study presented by Raissi et al. [29] uses Burger's equation for three possible applications: (1) solve a known PDE given initial and boundary conditions, (2) find parameters that govern a known PDE based on data from the objective space, and (3) find and solve an unknown PDE solely based on data from the objective space.

There are currently no PINN applications to PHM frameworks in the reliability community. This is mainly due to the lack of equations that can link a complex system's degradation dynamics with its condition monitoring sensor readings. Nevertheless, most DL-PHM frameworks seek to relate the monitoring variables with the system's diagnosis and prognosis. As such, the PINN approach proposed in [29] presents an opportunity to seek and find possible unknown PDEs that can relate sensor measurements to the system's degradation process.

3. Proposed Framework

Obtaining models that simultaneously yield an interpretable health estimator and traditional prognostics metrics is an ongoing challenge in DL-PHM models [2, 9]. An interpretable model allows the user to trust its prediction, which is critical for implementing DL-PHM models for the health management of real systems. Training a DNN to represent the degradation process in a complex system is difficult due to the lack of mathematical models to describe its physics of degradation. Moreover, most DL models applied to PHM do not consider time as an input variable of the network. Thus, information regarding the degradation dynamics of the system is lost during the training process if not explicitly stated (as in PBMs). In the case of RUL estimation, another challenge is presented when creating labels for supervised models. Here, it is common to define a point at which the degradation process starts. This can be either at a fixed time before failure [34] or when a specific performance variable surpasses a predefined threshold value [35]. Both approaches impose a strict constraint to the RUL labels by assuming that the machinery under study will continue to operate in the same condition until its failure. A DL model trained with these labels will inevitably be biased towards this behavior, making it susceptible to errors when tested with new data. Nevertheless, we can overcome this uncertainty by giving interpretability to our model.

Since there are no available equations to directly map the health state of a complex system to its operational conditions, we propose a DNN framework to explore the degradation physics of a system through a latent space representation. The supervised framework is aimed at PHM prediction tasks, where operational data is available from the monitoring of a system. The framework establishes a relationship between a latent variable and a prognosis output variable through a PDE-like penalization function (equation (8)). By training the DNN to understand the dynamics of the degradation process, it is expected that the model will improve its generalization capabilities. Indeed, adding a PDE-like penalization to the loss function of the model creates a relationship between the input features of the model and the derivatives of the output value with respect to its independent variables. This effect can be boosted if the framework is given time as an input feature, rather than implicitly extracting it from a sequence. For metrics such as the RUL, the penalization function adds information on the degradation rate by considering temporal derivatives.

Figure 1 illustrates the proposed DNN framework. It yields RUL estimations through three stages, represented by three different NNs. The first stage maps the operational conditions (OCs) and the time t to a (possibly multidimensional) latent variable x. A second NN then takes both t and x to yield the RUL estimation of the system. A third NN is used to model the right-hand side of equation (5) $F(z, u, u_z, \ldots)$, which models the RUL's time derivative through a NN. This is the so-called dynamics of the PDE. The NN for each stage of the proposed framework is structured as follows:

(1) x-NN: the network takes the OCs and time as input variables and it outputs the latent variable x. It is comprised of 5 hidden layers of 3 units each and two units as an output layer. This accounts for 104 trainable parameters. Hyperbolic tangent (tanh) is used as the activation function. The dimensionality of the latent variable is a hyperparameter that needs to be tuned according to the system under study.

(2) RUL-NN: it takes both the latent variable x and time as input values and outputs the RUL of the system. It comprises of 5 hidden layers of 10 units each to yield one output unit with tanh as the activation function. The network encompasses 481 trainable parameters.

(3) Dynamics-NN: it takes the latent variable x and the derivatives $\mathrm{d}x/\mathrm{d}t$ and $\mathrm{dRUL}/\mathrm{d}x$ as input values. It outputs a function N that represents the dynamics of the system. This goes into the PDE-like penalization function. The network is comprised of 5 hidden layers of 10 units each. One output unit and a rectified linear unit (ReLU) as the activation function are used. This network also contains 481 trainable parameters.

With automatic differentiation, we can take the time derivative from the first and second stage of NNs. These are then combined to form a PDE-like penalization term, as shown in Figure 1. The penalization includes the time derivative from the RUL, which is related to the Dynamics-NN in a PDE-like form as shown in equation (8), where $\mathrm{d}(\mathrm{RUL})/\mathrm{d}t$ is the time derivative of the second stage NN, and

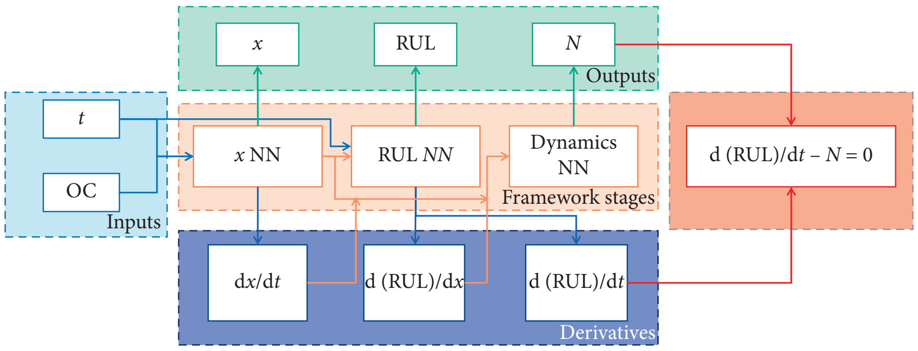


Figure 1: Proposed deep learning framework using a latent variable and a PDE-like penalization function.

N is the output from the third stage NN. The training cost function is then defined as shown in equation (9), where RUL_i and $\widehat{\mathrm{RUL}}_i$ are the objective and predicted RUL values, respectively:

$$f: \frac{\mathrm{d(RUL)}}{\mathrm{d}t} - N\left(x, \frac{\mathrm{d}x}{\mathrm{d}t}, \frac{\mathrm{d(RUL)}}{\mathrm{d}x}\right), \quad (8)$$

$$\mathrm{Cost} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathrm{RUL}_i - \widehat{\mathrm{RUL}}_i\right)^2 + \lambda\frac{1}{N}\sum_{j=1}^{N} f^2. \quad (9)$$

The penalization function f thus creates a dynamical relationship between the RUL and the latent variable x, which in turn is related to the initial operating conditions and the time at which the RUL is evaluated. The framework is comprised of 1,066 trainable parameters, which is a low number when compared to other significantly more complex DL architectures for RUL estimation [36]. Having a model with fewer parameters to train prevents overfitting and reduces the training time, which can eventually facilitate its online implementation without the need for specialized hardware.

The proposed framework addresses many of the drawbacks mentioned above in DL applications for PHM. First, the network takes time as an input variable, along with the operational conditions of the system. The OCs represent the initial conditions for a PDE, and t corresponds to the point in the future at which it is desired to obtain the RUL value. In other words, for $t = 0$, the network behaves as most DL methods. That is, RUL is predicted based on the current OCs. Secondly, the use of a latent variable provides multiple advantages for both the training of the model and the later interpretation of its results:

(1) Dimensionality reduction: the usage of a latent variable helps capture and highlight important information related to the degradation process from the OCs. The dimensions of the latent variable dictate the number of dimensions that we can use for visualization purposes. In turn, visualizing a latent space provides additional tools to make an informed decision based on the model's output.

(2) Input variables for Dynamic-NN: the right-hand side function in equation (6) could take every possible derivative from the input OC values. The use of a latent variable reduces the number of derivatives fed into the Dynamics-NN, thus reducing the number of parameters of the network and its training time.

(3) Eliminate redundancy and noise from the data: due to the potentially high correlation among monitoring variables, it is common to observe that a lower-dimensional space can represent a system. This is the basic concept behind every data-driven approach for regression in PHM. Further, DNNs are known to remove noise levels in the input signals.

Note that out of the three stages, only the RUL-NN requires labels for the training process, since the latent variable x comes as a secondary outcome from the backpropagation training of the RUL-NN. On the other hand, the Dynamics-NN is trained solely from the penalization PDE term, which does not require any labels. Furthermore, if a degradation equation is available, for example, Paris' Law for crack propagation, it can be directly replaced for the Dynamics-NN, giving our proposed model flexibility according to the available information on the system under study.

To train models based on the proposed framework, the following steps must be followed:

(1) Preprocess the dataset. The input data to the framework has two essential elements: sensor measurements and prediction time horizon. Details on the dataset preparation are presented in Section 4. Given that this is a supervised framework, objective labels associated with the input values must also be provided.

(2) Define and set up the framework (Figure 1) according to the available data and information on the system under study. If available, a PBM (e.g., Paris' Law) can be included in the penalization function, replacing the Dynamic-NN. Otherwise, the Dynamic-NN is used to discover the system's degradation dynamics. Other hyperparameters such as the dimensionality of the latent variable x, the number of neurons and layers of each NN, and the penalty weight λ need to be selected as discussed in Section 5.

(3) Train the model based on the chosen framework with the preprocessed dataset. All NN stages within the

framework are trained simultaneously according to equation (9).

(4) Once the model is trained and depending on the selected dimensionality of the latent variable x, results can be visualized by evaluating new input values and plotting each dimension of x on a different axis. Here, the output values of the model (e.g., RUL) are used as a color map. This visualization allows directly assessing the relationship between the trained latent variable and the objective function. As discussed in Section 5, the latent variable can be used as a health estimator in the PHM context through a ML classifier.

4. Case Study: Dataset and Hardware

The proposed framework is tested using the benchmark dataset C-MAPSS due to the multiple research reports that have applied DL networks for its RUL estimation [6, 20, 24, 37, 38]. A detailed description of this dataset and its processing can be found in [39, 40]. This study's objective is not to improve the state-of-the-art results on this dataset regarding RUL estimation precision, but rather to introduce a new tool for PHM-DL models. Hence, only the FD001 and FD004 subdatasets from the C-MAPSS will be covered in detail. The dataset consists of 27 sensor variables for simulated engine runs. The FD001 dataset presents one failure mode and one operational condition. FD004 on the other hand presents two different failure modes, and it operates at six different conditions. The information on which failure mode caused the failure of the engine run is not provided with the dataset, nor are the conditions at which they were operating before the failure. Operational sensor readings are recorded for each cycle during an engine run. Each engine run starts at a random initial degradation level from which the engine operates until its failure.

As has been shown in past studies [40], only 14 out of the 27 sensors are statistically significant to model the RUL of the system, and thus these are the ones used for this study. Since the proposed framework is based on vanilla DNNs, there is no need to create time windows for the input data. However, we need to create a temporal dimension (i.e., feature) in order to train the proposed model. As such, the original dataset needs the following additional processing steps:

(1) Select all data logs for one engine run, from its initial starting point until its failure.

(2) For each operational cycle, add a column with an integer time t from 0 to 30 cycles.

(3) Create a label for the above operation data and time, which corresponds to the RUL value at time t since the initial point.

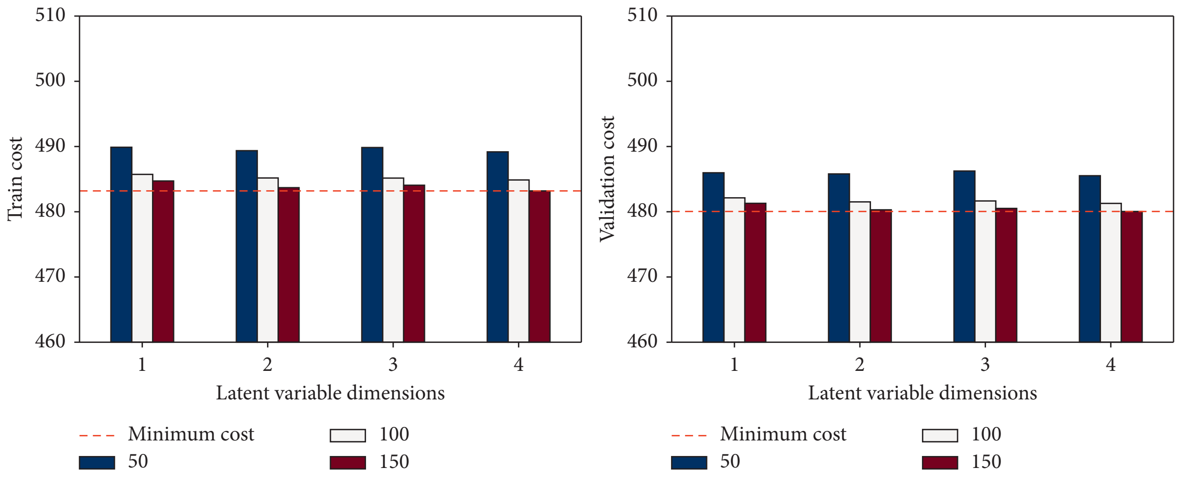
For instance, let us consider Engine 1, which contains a total of 192 log entries. If cycle number 100 is selected as the initial point, then, for $t = 0$, its corresponding label is RUL = 92; then, for $t = 1$, its label is RUL = 91; and so on until $t = 30$ is reached or until RUL = 1 (i.e., the engine fails). This process is repeated for each log entry of each engine, which increases the size of the original dataset. For instance, the FD001 subdataset size increases from 20,631 to 593,061 points. The input values are normalized using a MinMax scaler, which is a common practice when training DNNs [41]. Models are trained on Python 3.6 with the use of Tensorflow 2.0 and Keras. Windows is used as the operating system. The computer hardware consists of an Intel i7-9700k CPU, 32 GB of RAM memory, and a 24 GB Titan RTX GPU. The average training time in this machine is 140 seconds, while the evaluation time for new data entries is 0.01 seconds.

The value range of the newly added time feature column is an additional hyperparameter of the proposed framework. This will depend on the specific system under study, and in this case, it was selected based on the following reasoning. The time horizon for RUL estimation needs to be realistic. In this regard, if a system begins operating from an almost perfect health state, there would not be an indication of the degradation process within the monitoring data. Hence, it would be optimistic to expect the model to accurately estimate future RUL values at times close to the end of the system's life based on this data. As such, we should not train the model to yield RUL predictions at times exceeding the training RUL labels values. Since the RUL labels for the C-MAPSS range from 1 to 125 cycles, the time dimension should at most range from 0 to 125 cycles. Based on this reasoning, we tested the framework with prediction horizons from 0 to 100 cycles. We observed that the model's performance decreases significantly for horizons greater than 30 cycles. Thus, we chose this as the upper time limit, which accounts for almost one-quarter of the training label range.

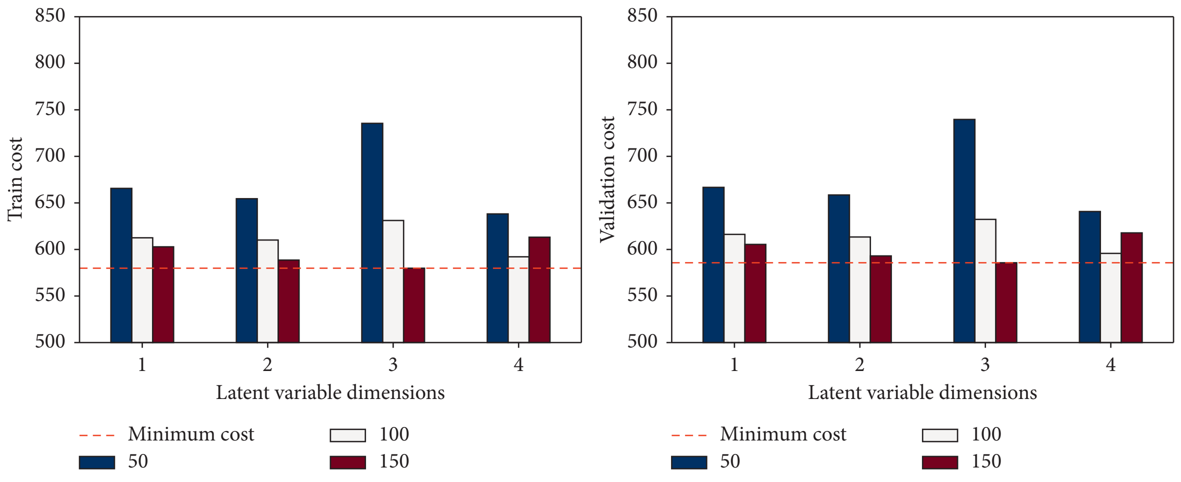
5. Results and Discussion

We train the proposed framework for the FD001 and FD004 subdataset from C-MAPSS. Models are trained using 75% of the data randomly selected from the training set, with the remaining 25% left as a validation set. The test sets are provided separately [40]. NAdam optimizer [42] is used for the training process. The proposed framework comprises multiple hyperparameters; three of these have the most significant impact on the model's performance after training: the latent variable dimensionality, the penalty weight λ assigned to the PDE regularization function, and the number of training epochs. Figures 2 and 3 illustrate the results for the sensibility analysis of these three hyperparameters. For each combination of hyperparameters, 10-fold cross-validation was performed with random initial parameters. We compare the average cost function value on the training and validation set from the cross-validation process in these figures. The minimum cost is indicated with a red dashed horizontal line for each case.

Figure 2 shows the joint sensibility analysis for the number of training epochs and the latent variable dimension. On one hand, most cost values decrease with a higher number of training epochs for both the training and the validation set as expected. This behavior is shown by both subdatasets, independent of their complexity. On the other



(a)



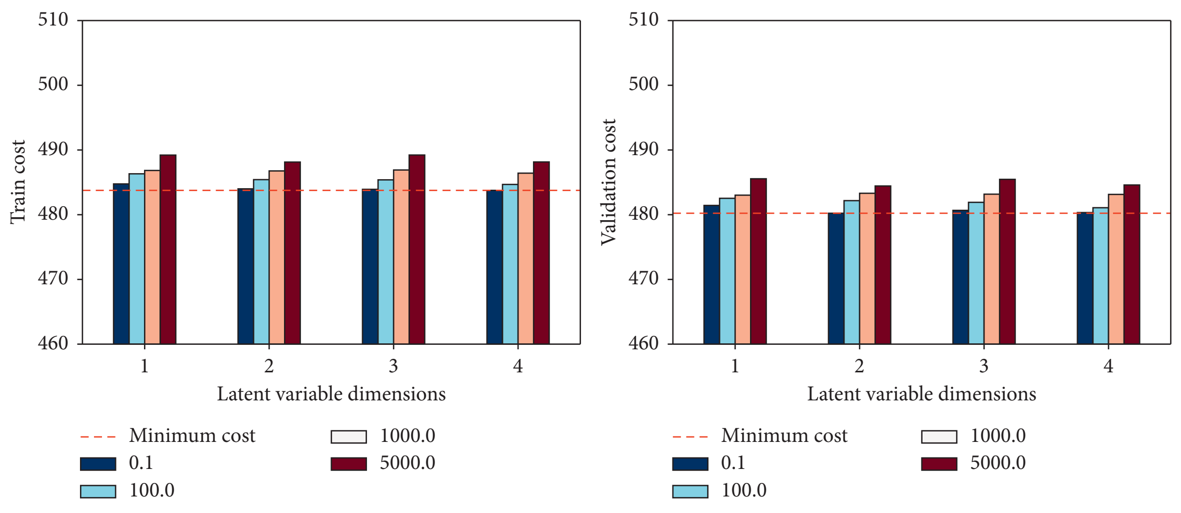
(b)

Figure 2: Sensibility analysis of the cost function by the number of training epochs. (a) FD001. (b) FD004.

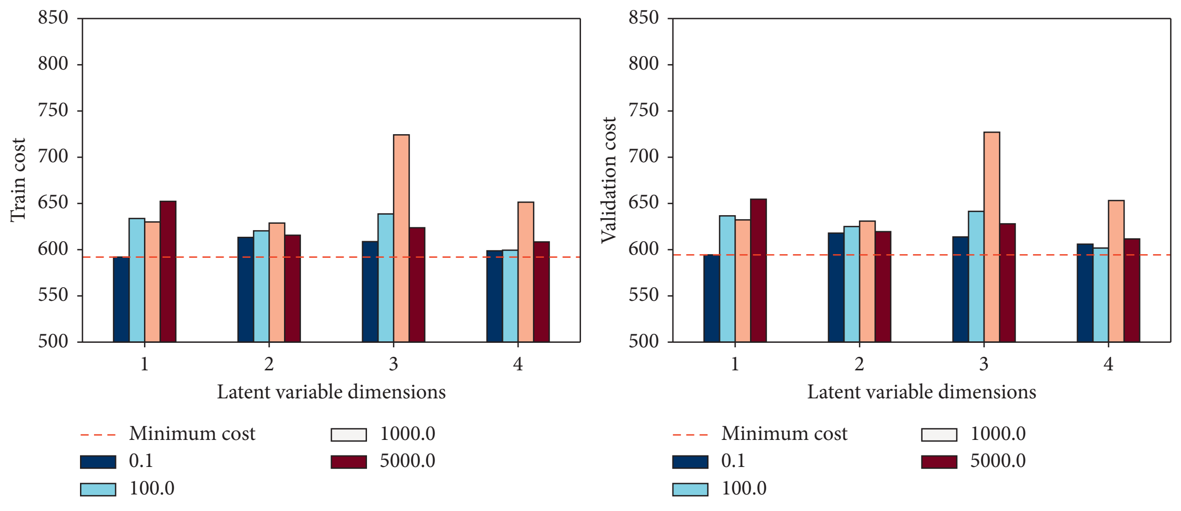
hand, we can see that the best results are achieved with a two-dimensional latent variable on the FD001 set, while the FD004 set performs better with a three-dimensional latent variable. These results are consistent with the complexity difference between the datasets since the FD004 set contains six operational conditions and two failure modes. Thus, the model requires a higher latent variable dimensionality to represent the degradation process. Further, the results shown in Figure 2 for the FD001 set indicate that models have similar performance for a latent variable with more than two dimensions. In the case of the FD004 set, a similar performance is obtained for two and three dimensions.

Figure 3 presents the joint sensibility analysis for the PDE penalization weight value and latent variable dimension. The penalization function improves the generalization capabilities of the model, resulting in similar cost performance when evaluating the training and validation sets. However, the specific value of the weight penalization is the most difficult hyperparameter to analyze. A higher penalization value results in a more constrained model, and thus, its performance worsens when evaluating the training set. For instance, we can observe that, with a low penalization value, the model presents underfitting (i.e., the validation cost value is lower than the training) on the FD001 set regardless of the latent variable dimensionality. Nevertheless, a higher penalty value during the training process would give higher importance to the connection between the latent variable representation and the RUL of the system. This would explain the more consistent behavior between the training and validation set for the more complex FD004 set. Both datasets have a consistent cost value with higher penalization weights in the case of a two-dimension latent variable, particularly the FD004 set, where there is neither significant underfitting nor overfitting. Hence, a two-dimensional latent variable is better when considering the PDE penalization function.

From this hyperparameter analysis, a two-dimensional latent variable is selected due to its more consistent results. This is also a good choice for visualization purposes, given that one will be able to map all the dimensions in a two-dimensional latent space representation once the model is trained. Moreover, models are trained for 150 epochs and a



(a)



(b)

Figure 3: Sensibility analysis of the cost function by PDE penalty weight value λ. (a) FD001. (b) FD004.

penalty weight value of 100. Ten different models are trained for each dataset, each starting from random initial weights for the three NN stages.

Table 1 presents the results for the average root mean squared error (RMSE) obtained with the trained models for each dataset. FD001 models average an RMSE value of 17.14 cycles for its test set, while models for FD004 yield an average of 25.58 cycles. Figure 4 illustrates the training and validation cost throughout the training process. Here, it can be observed that both curves present an identical behavior. Also, these converge to the same cost value and, thus, the trained models have good generalization capabilities. We can attribute this behavior to the PDE penalization function added to the model. The dynamical relationship built between the latent variable and the RUL, as well as the inclusion of the time dimension, provides extra information on the degradation dynamics to the model. In turn, the model can yield consistent predictions for new unseen data. The behavior of the cost function during the training process is also consistent with the hyperparameters effects studied in Figures 2 and 3.

Table 1: Training and testing RUL RMSE values for models trained based on the proposed framework.

	Training RMSE	Test RMSE
FD001	21.96	17.14
FD004	24.72	25.58

Although the obtained RMSE values for the test sets are not as low as those obtained through other far more complex architectures, these are within the acceptable range for this case study [20]. Such complex architectures involve a high number of trainable parameters without providing interpretability tools for the end-user. For instance, a deep convolutional neural network (DCNN) for RUL estimation with over 180k trainable parameters was presented in [20]. This number increases further when additional layers of analysis are added to CNNs, such as multiscale blocks [43] or bidirectional LSTMs [44]. They require more preprocessing for their training data and can present overfitting while requiring specialized hardware for a fast training process.

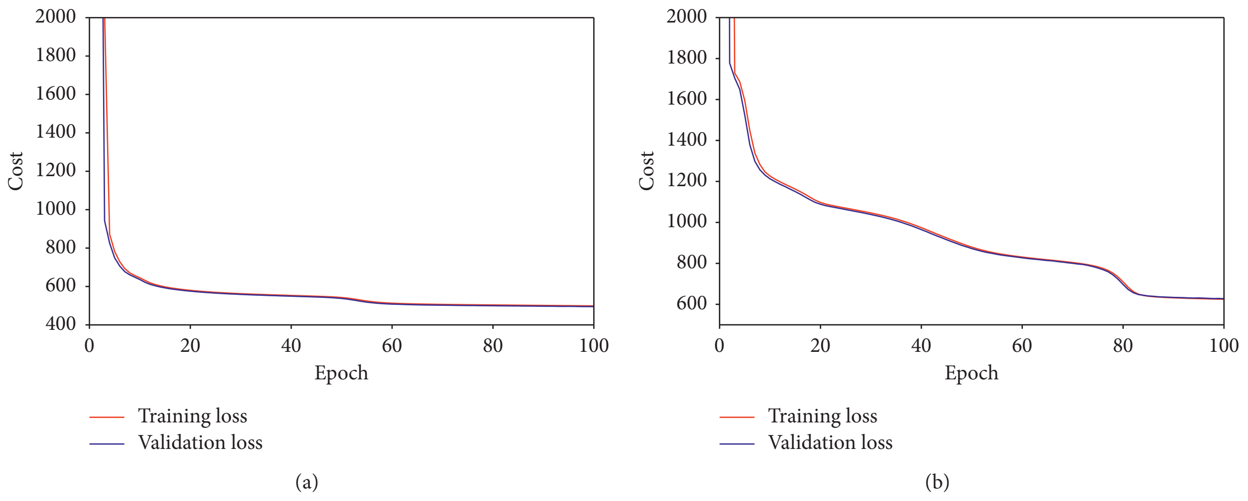


Figure 4: Training and validation cost value per number of epochs during the training process for the FD001 (a) and FD004 (b) subdatasets.

Additionally, the more trainable parameters a model has, the more training data is needed to prevent overfitting.

The proposed framework's most important output is the latent space representation of the trained models. Indeed, Figure 5 illustrates the predicted RUL values mapped to their corresponding estimated latent variable space for both the training and test sets. Both dimensions from the latent variable x (i.e., x_1, x_2) are plotted with their corresponding RUL values represented as a color map. Figure 5 shows three different RUL mappings for each subdataset. On the left, the RUL training labels are mapped to their corresponding latent space representation. At the center, RUL values predicted by the trained model evaluated with the training set (i.e., same input data as the previous case) are mapped to their corresponding latent space. On the right, similar to the first figure, an RUL colormap is presented for the latent space representation of the test set labels.

Results presented in Figure 5 for the training set show that the trained model smooths the RUL value representation to the latent space. This creates a continuous relationship between the operational conditions and the RUL of the system. Given the linear relationship between the RUL and time, a health index related to the RUL is analogous to an indicator of the temporal evolution of the system's degradation process. Hence, we can consider the latent space representation in Figure 5 as a health state indicator related to the system's underlying degradation process. Moreover, Figure 5 shows that both subdatasets present different shapes on their latent space representation. This is expected since both datasets present a different number of failure modes. Indeed, given that the FD001 set has only one failure mode, a latent space domain following a straight-line path from low to high RUL values makes us think this is a good representation of the degradation process of this particular system. This degradation path is also simpler than its FD004 counterpart. In the latter, we see that, from a healthy state (i.e., high RUL values), the latent space presents a bifurcation into two degradation paths. Since this dataset comes from a system with two different failure modes, we believe these degradation paths can be the model's interpretation of the failure modes. Unfortunately, information on which failure mode caused the system's failures is not available to confirm this observation.

In the case of the test set representation from each subdataset in Figure 5(a), we observe that both the RUL mapping and the shape of the latent space representation of the test set are consistent with those obtained for the training set (center images). This reinforces the generalization capabilities of the models discussed in Figure 4, where we observe that the training and validation cost curves were almost identical throughout the training process. These results from the test sets indicate that the latent space can indeed be used as an indicator of the system's health state. This interpretability is why including time as an input variable becomes crucial to our proposed framework. By including time, it is possible to obtain the temporal derivative of the RUL (i.e., the RUL dynamics), which defines the PDE penalization function. This, as Figure 5 shows, allows us to embed the degradation process to the evolution of the RUL values along with the latent space representation. In both subdatasets, by considering the transition from high RUL values into lower ones as a temporal evolution of a health index, we can use the latent space to determine the health state of the system if it were to be separated by an RUL threshold.

Indeed, using the training dataset, we define a "start of degradation" threshold to separate the health state of the system as either "healthy" or "degraded." Having two different classes allows us to train machine learning classification models based on the results obtained for the latent space representation. This threshold (TH) value needs to be optimized to provide an accurate classifier and ensure that the degradation detection occurs with enough anticipation of the failure. For instance, a TH of 20 cycles considers all points with $\mathrm{RUL} \leq 20$ cycles as degraded, while for $\mathrm{RUL} > 20$ cycles, the system is considered as healthy. However, 20 cycles before failure would not be reasonable since it is too close to the failure event. A TH value of 120, on

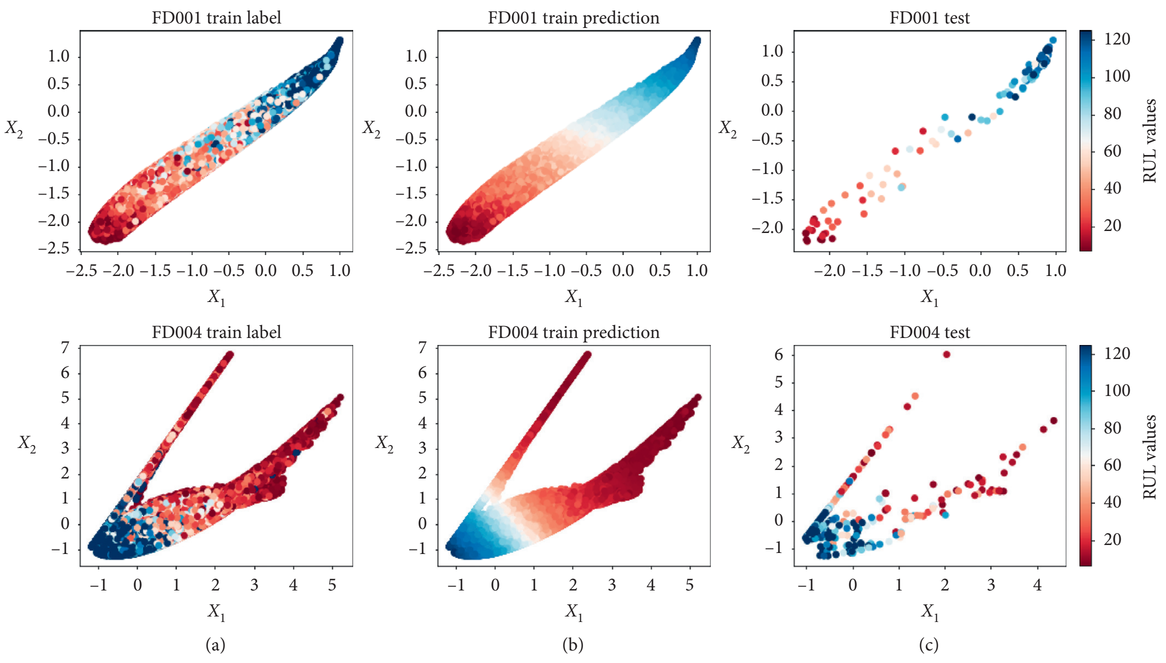


Figure 5: Trained latent variable mapping of RUL training labels (a), RUL training data predictions (b), and RUL testing labels (c).

the other hand, would not be informative since all the RUL labels from the C-MAPSS dataset are commonly defined as lower or equal to 125 cycles. As such, a sensibility analysis to select the TH value is needed for different classifiers. Here, we will focus on six of the most common classifiers in ML approaches. The classifier and TH selection would depend on the accuracy of the model on both the training and validation sets, as well as on the specific system under study.

Figure 6 shows the results for the TH and ML classifier sensibility analysis. All classifiers are trained using the default parameters provided in the sci-kit learn package for Python [45]. The analysis is performed over both subdatasets individually, and results are reported for the training and validation sets. It can be observed that the classification performance decreases with higher TH values for the training and validation sets.

As discussed above, setting a small TH value for the classifier can be impractical from a PHM perspective. Hence, we select 50-cycle TH value for being the longest horizon where classifiers present a 90% accuracy performance while still accounting for 40% of the RUL label range. Tables 2 and 3 detail the performance metrics for all classifiers at TH = 50. Here, the best training performance is obtained with a Nearest Neighbors classifier, which has the lowest validation accuracy and overfits the training set. All remaining classifiers present a similar performance on the training and validation sets with no overfitting. This behavior was also observed in Figure 6. Random forest (RF) stands out among these classifiers due to its low training time and false positive metric. RF is also known for providing good visualization representation that allows us to separate classes visually. As such, we select RF as the classifier for the health state estimator through the latent space representation.

Table 4 presents the results after training ten RF classifiers for each subdataset. Results show a high accuracy for both subdatasets, with all false negatives and false positives below 10%. The FD001 set presents a slight underfitting of its results, which can be associated with the great number of training points generated to include the time dimension during the training process, as discussed in Section 4.

Figure 7 illustrates the trained RF classifier results for both the training and test sets mapped to the latent space representation. Observe the classifier clearly separates a healthy zone (blue) and a degraded zone (red). This classifier is fairly conservative, especially for the FD004 subdataset, mainly due to the selected threshold. It is important to note that, for both subdatasets, the mapping of the test set is consistent with the trained classifier and the RF classifier provides a transition zone (white) which works as a separation boundary between the two defined health states. The trained classifier is not limited to only two classes, and more health states could be added if they are available or needed. These results show that the latent space representation can indeed be used as a health indicator and, as such, can work as a decision variable when planning maintenance policies.

As shown in Figure 5, the shape of the latent variable representation changes from one subset to another. It is then logical that if the proposed framework were to be tested for another system, the visualization and RUL mapping would also be different from the case study discussed. As such, training the selected classifier and setting the corresponding threshold value would vary from system to system. Also, depending on the PHM framework and implementation, setting an optimal threshold value may also depend on other metrics rather than just the models' efficiency. For instance, an optimized TH with a classifier accuracy of 90% might be worth more than having a small TH value (e.g., 5 cycles

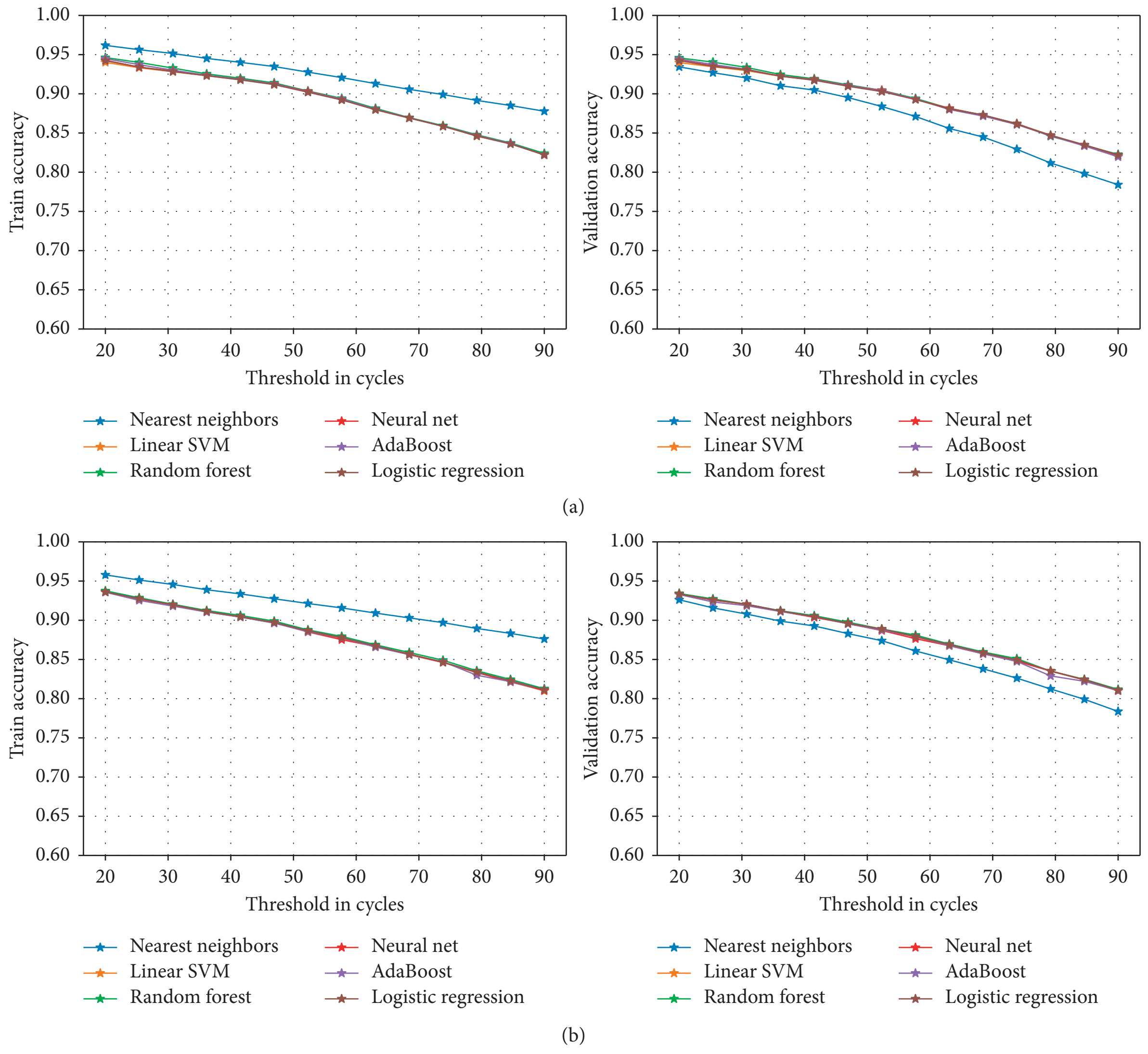


Figure 6: Threshold sensibility analysis for six machine learning classifiers for the health state of the system based on the trained latent variable x. (a) FD001. (b) FD004.

Table 2: Machine learning classifiers performance with TH = 50 for the FD001 subdataset.

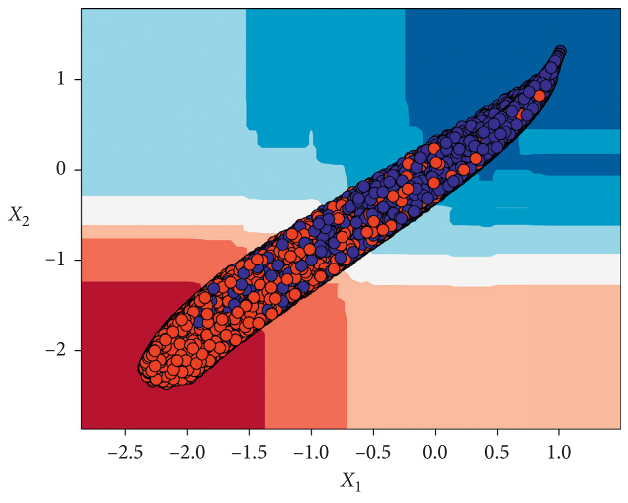
ML classifier	Set	Accuracy	False negative	False positive	$F1$-score	Recall	Training time
Nearest neighbors	Train	93.0	4.6	2.4	86.3	82.7	3.57
	Validation	88.7	6.7	4.6	78.1	75.0	
Linear SVM	Train	90.6	6.2	3.2	81.3	76.8	69.37
	Validation	90.7	6.1	3.2	81.6	77.2	
Random forest	Train	90.7	6.7	2.6	81.1	74.9	0.81
	Validation	90.6	6.7	2.7	81.0	75.1	
Neural network	Train	90.6	6.1	3.3	81.3	77.1	10.43
	Validation	90.7	6.0	3.3	81.7	77.5	
AdaBoost	Train	90.6	6.5	3.0	81.0	75.7	3.80
	Validation	90.5	6.5	3.0	81.1	75.9	
Logistic regression	Train	90.5	6.0	3.5	81.3	77.6	1.44
	Validation	90.6	5.9	3.5	81.6	77.9	

Table 3: Machine learning classifiers performance with TH = 50 for the FD004 subdataset.

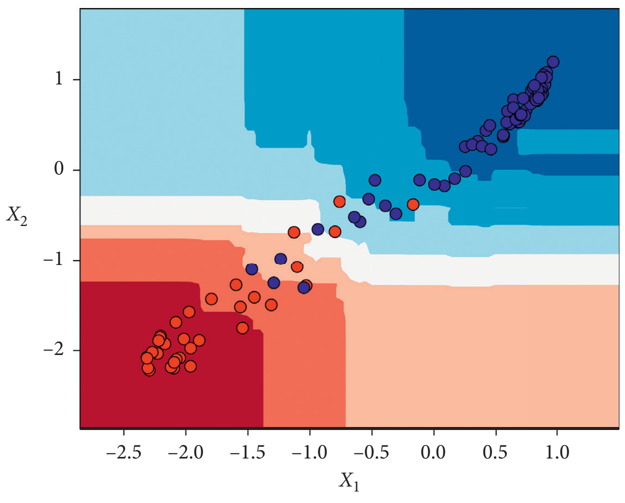
ML classifier	Set	Accuracy	False negative	False positive	*F*1-score (%)	Recall (%)	Training time (s)
Nearest neighbors	Train	92.4	4.9	2.7	81.9	77.8	4.02
	Validation	87.8	7.2	5.0	71.2	67.7	
Linear SVM	Train	89.0	8.3	2.7	71.3	62.3	94.96
	Validation	89.1	8.2	2.6	72.1	63.0	
Random forest	Train	89.1	7.9	3.0	72.1	64.0	0.98
	Validation	89.2	7.9	2.8	72.8	64.5	
Neural network	Train	89.0	7.5	3.5	72.6	66.0	9.31
	Validation	89.2	7.5	3.3	73.3	66.5	
AdaBoost	Train	88.8	7.8	3.5	71.7	64.7	4.82
	Validation	88.9	7.8	3.3	72.4	65.1	
Logistic regression	Train	89.0	8.0	2.9	71.7	63.4	1.53
	Validation	89.2	8.0	2.8	72.6	64.1	

Table 4: Classification metrics for random forest models with a 95% confidence interval.

		Accuracy (%)	False negative (%)	False positive (%)	*F*1 score (%)	Recall score (%)
FD001	Train	90.7 ± 0.0	6.6 ± 0.2	2.6 ± 0.2	81.1 ± 0.1	74.9 ± 0.8
	Test	92.9 ± 91.2	3.1 ± 0.6	4.0 ± 0.6	89.4 ± 1.8	90.6 ± 1.8
FD004	Train	89.1 ± 0.2	7.8 ± 0.2	2.9 ± 0.2	72.3 ± 0.3	64.3 ± 0.9
	Test	86.7 ± 0.3	9.1 ± 0.3	4.1 ± 0.4	77.5 ± 0.5	71.5 ± 0.9



(a)



(b)

Figure 7: Continued.

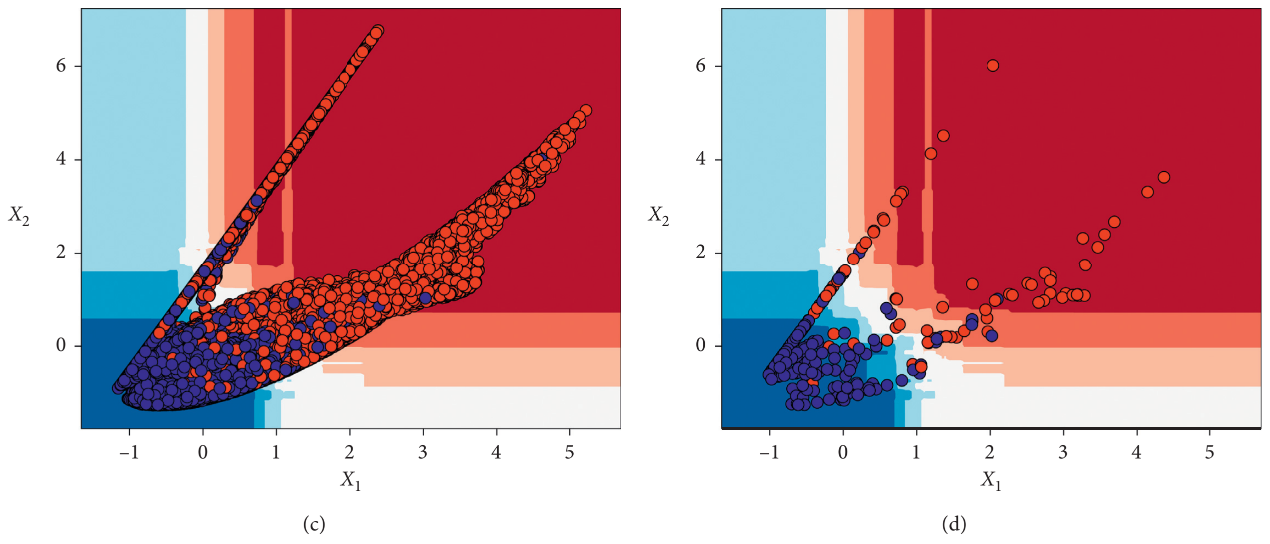


Figure 7: Latent variable classifier decision zones based on a trained random forest model for the FD001 and FD004 subdatasets: degraded state (red) and healthy state (blue). (a) FD001 train. (b) FD001 test. (c) FD004 train. (d) FD004test.

before failure) with a 99% classifier accuracy. An online implementation of the model, along with the classifier, would allow a real-time evaluation of the system's operational conditions. This classifier can be further complemented with the remaining stages of the framework presented in Figure 1, that is, the PDE dynamics N and the RUL estimation. These additional outputs provide information on the system and can be used to create new metrics, rather than just base the results on an RUL value. Thus, this framework creates the opportunity to make better-informed decisions for the maintenance scheduling of complex systems.

Recent DL research works using the C-MAPSS dataset as a case study have focused on feature extraction to improve models' performance, supervised health state estimation, and optimal RMSE values. For instance, Berghout [46] used a denoising autoencoder as a feature extractor coupled with an update selection strategy to determine the training sequences used in an extreme learning machine (ELM) prognosis model. Here, only the FD001 subdataset was trained. Due to the feature extraction process and the ELM prognosis model, this framework contains a high number of trainable parameters and its good performance is likely to be case specific. This model does not provide classification nor visual interpretation. Another example is presented by [30] where a multitask deep CNN is proposed for simultaneous health state and RUL estimation. This model requires manually creating health state labels, which introduces bias to the model, and it does not provide any interpretation of the model. The dual estimation also increases the number of trainable parameters significantly. Results for these configurations and other traditional DL applications to PHM are compared in Table 5. Up to date, there are no frameworks that can provide prediction, classification, and visual representation at the same time. The lower performance on RUL estimation could then be viewed as a tradeoff between

Table 5: Comparison with the state-of-the-art results for RUL RMSE values and state of health classification accuracy for the FD001 and FD004 test sets.

	FD001		FD004	
Model	RMSE	Accuracy	RMSE	Accuracy
ANN	19.62	—	24.35	—
RNN	13.36	—	24.06	—
DOS-ELM [46]	12.29	—	—	—
MT-CNN [30]	12.48	0.71	19.98	0.84
PDE-PHM	17.14	0.92	25.58	0.86

prognosis and interpretability of the model. Furthermore, the fewer parameters of our proposed model avoid overfitting problems and, as it was discussed, adapting the framework to other case studies is straightforward. Here, it is important to remark that the proposed framework can be adapted to consider a PBM, when available.

6. Conclusions

This paper presented a framework with the first application of PINN applied to PHM in complex systems. The proposed framework allows the interpretation of the degradation dynamics through a latent space representation and, thus, it is a promising alternative for physics-informed model applications for complex systems. The framework comprises deep neural networks with a total of 1,066 parameters, which is considerably smaller than more complex architectures by at least two orders of magnitude. This contributes to low training and evaluation times while preventing overfitting and makes it a suitable approach to be deployed both online and on mobile devices. This framework establishes a relationship between the time and sensor variables with the degradation of a PDE-like penalization function. We have shown that the obtained two-dimensional latent space acts as a health indicator of the degradation process of the system, which also can be visually interpreted for engineering purposes as well as a health state classifier through an ML model. Additionally, the proposed framework addresses two major challenges in DL techniques applied to PHM, namely, the use of time as an input variable and the interpretation of the operational conditions from an engineering point of view. This paper takes a step towards bridging the gap between statistic-based PHM and physics-based PHM by providing models that do not need ad hoc and third-party software to interpret its results and it is directly linked to the degradation process of the system. The presented framework is flexible because it can integrate available degradation processes into the training process if these are available. The framework opens many doors to applying these algorithms to real complex systems, especially on maintenance and preventive assessments.

Data Availability

For this study, the C-MAPSS dataset was used. The dataset is publicly available at https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/.

Conflicts of Interest

The authors declare that there are no conflicts of interest.

Acknowledgments

This work was carried out as a part of the Pipeline System Integrity Management Project, which was supported by the Petroleum Institute, Khalifa University of Science and Technology, Abu Dhabi, UAE. Sergio Cofre-Martel would like to thank the Agencia Nacional de Investigacion y Desarrollo (ANID/Conicyt) for its support through the Becas de Doctorado en el Extranjero, Becas Chile Program.

References

[1] Y. Lei, N. Li, L. Guo, N. Li, T. Yan, and J. Lin, “Machinery health prognostics: a systematic review from data acquisition to RUL prediction,” *Mechanical Systems and Signal Processing*, vol. 104, pp. 799–834, 2018.

[2] B. Rezaeianjouybari and Y. Shang, “Deep learning for prognostics and health management: state of the art, challenges, and opportunities,” *Measurement*, vol. 163, Article ID 107929, 2020.

[3] A. Cubillo, S. Perinpanayagam, and M. Esperon-Miguez, “A review of physics-based models in prognostics: application to gears and bearings of rotating machinery,” *Advances in Mechanical Engineering*, vol. 8, no. 8, pp. 1–21, 2016.

[4] Y. Li, K. Liu, A. M. Foley et al., “Data-driven health estimation and lifetime prediction of lithium-ion batteries: a review,” *Renewable and Sustainable Energy Reviews.*vol. 113, 2019.

[5] L. Liao and F. Köttig, “Review of hybrid prognostics approaches for remaining useful life prediction of engineered systems, and an application to battery life prediction,” *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 191–207, 2014.

[6] A. Al-Dulaimi, S. Zabihi, A. Asif, and A. Mohammadi, “A multimodal and hybrid deep neural network model for remaining useful life estimation,” *Computers in Industry*, vol. 108, pp. 186–196, 2019.

[7] L. Liao, W. Jin, and R. Pavel, “Enhanced restricted Boltzmann machine with prognosability regularization for prognostics and health assessment,” *IEEE Transactions on Industrial Electronics*, vol. 63, no. 11, pp. 7076–7083, 2016.

[8] X. Li, W. Zhang, and Q. Ding, “Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction,” *Reliability Engineering System Safety*, vol. 182, pp. 208–218, 2019.

[9] O. Fink, Q. Wang, M. Svensén, P. Dersin, W.-J. Lee, and M. Ducoffe, “Potential, challenges and future directions for deep learning in prognostics and health management applications,” *Engineering Applications of Artificial Intelligence*, vol. 92, Article ID 103678, 2020.

[10] J. Ma, S. Xu, Y. Ding et al., “Cycle life test optimization for different Li-ion power battery formulations using a hybrid remaining-useful-life prediction method,” *Applied Energy*, vol. 262, Article ID 114490, 2020.

[11] Y. Zhou, M. Huang, and M. Pecht, “Remaining useful life estimation of lithium-ion cells based on k-nearest neighbor regression with differential evolution optimization,” *Journal of Cleaner Production*, vol. 249, Article ID 119409, 2020.

[12] G. Ma, Y. Zhang, C. Cheng, B. Zhou, P. Hu, and Y. Yuan, “Remaining useful life prediction of lithium-ion batteries based on false nearest neighbors and a hybrid neural network,” *Applied Energy*, vol. 253, Article ID 113626, 2019.

[13] X. Qiu, W. Wu, and S. Wang, “Remaining useful life prediction of lithium-ion battery based on improved cuckoo search particle filter and a novel state of charge estimation method,” *Journal of Power Sources*, vol. 450, Article ID 227700, 2020.

[14] Y. Chen, G. Peng, Z. Zhu, and S. Li, “A novel deep learning method based on attention mechanism for bearing remaining useful life prediction,” *Applied Soft Computing*, vol. 86, Article ID 105919, 2020.

[15] L. Xu, P. Pennacchi, and S. Chatterton, “A new method for the estimation of bearing health state and remaining useful life based on the moving average cross-correlation of power spectral density,” *Mechanical Systems and Signal Processing*, vol. 139, Article ID 106617, 2020.

[16] J. Zhu, N. Chen, and C. Shen, “A new data-driven transferable remaining useful life prediction approach for bearing under different working conditions,” *Mechanical Systems and Signal Processing*, vol. 139, Article ID 106602, 2020.

[17] Y. Wu, M. Yuan, S. Dong, L. Lin, and Y. Liu, “Remaining useful life estimation of engineered systems using vanilla LSTM neural networks,” *Neurocomputing*, vol. 275, pp. 167–179, 2018.

[18] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, “Long short-term memory network for remaining useful life estimation,” in *Proceedings of the 2017 IEEE International Conference on Prognostics and Health Management*, Dallas, TX, USA, June 2017.

[19] Z. Zhao, B. Bin Liang, X. Wang, and W. Lu, “Remaining useful life prediction of aircraft engine based on degradation pattern learning,” *Reliability Engineering & System Safety*, vol. 164, no. 457, pp. 74–83, 2017.

[20] X. Li, Q. Ding, and J. Q. Sun, “Remaining useful life estimation in prognostics using deep convolution neural networks,” *Reliability Engineering and System Safety*, vol. 172, pp. 1–11, 2018.

[21] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should i trust you?’ Explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD International Conference*

on Knowledge Discovery and Data Mining, New York, NY, USA, August 2016.

[22] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the Advances in Neural Information Processing System*, Long Beach, CA, USA, December 2017.

[23] S. Bach, A. Binder, G. Montavon, F. Klauschen, K. R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS One*, vol. 10, no. 7, Article ID e0, 2015.

[24] H. Miao, B. Li, C. Sun, and J. Liu, "Joint learning of degradation assessment and RUL prediction for aeroengines via dual-task deep LSTM networks," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 9, pp. 5023–5032, 2019.

[25] Y. Yu, C. Hu, X. Si, J. Zheng, and J. Zhang, "Averaged Bi-LSTM networks for RUL prognostics with non-life-cycle labeled dataset," *Neurocomputing*, vol. 402, pp. 134–147, 2020.

[26] A. Elsheikh, S. Yacout, and M. S. Ouali, "Bidirectional handshaking LSTM for remaining useful life prediction," *Neurocomputing*, vol. 323, pp. 148–156, 2019.

[27] L. Bellani, M. Compare, P. Baraldi, and E. Zio, *Towards Developing a Novel Framework for Practical PHM: A Sequential Decision Towards Developing a Novel Framework for Practical PHM: a Sequential Decision Problem solved by Reinforcement Learning and Artificial Neural Networks*, 2020.

[28] Z. Mahmoodzadeh, K. Wu, E. L. Droguett, and A. Mosleh, "Condition-based maintenance with reinforcement learning for dry gas pipeline subject to internal corrosion," *Sensors*, vol. 20, no. 19, pp. 1–31, 2020.

[29] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[30] T. S. Kim and S. Y. Sohn, "Multitask learning for health condition identification and remaining useful life prediction: deep convolutional neural network approach," *Journal of Intelligent Manufacturing*, pp. 1–11, 2020.

[31] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of the COMPSTAT'2010*, Paris, France, August 2010.

[32] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Proceedings of the International 1989 Joint Conference on Neural Networks*, Washington, DC, USA, June 1989.

[33] A. Otto, K. Griewank, and A. Griewank, "On automatic differentiation automatic/algorithmic differentiation view project abs-linear learning by gradient based methods or mixed binary linear optimization view project on automatic diierentiation 1 by on automatic diierentiation," 1997, https://arxiv.org/abs/1502.05767.

[34] M. W. M. G. Dissanayake and N. Phan-Thien, "Neural-network-based approximations for solving partial differential equations," *Communications in Numerical Methods in Engineering*, vol. 10, no. 3, pp. 195–201, 1994.

[35] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[36] J. Chen, H. Jing, Y. Chang, and Q. Liu, "Gated recurrent unit based recurrent neural network for remaining useful life prediction of nonlinear deterioration process," *Reliability Engineering & System Safety*, vol. 185, pp. 372–382, 2019.

[37] A. Listou Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, and H. Zhang, "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture," *Reliablility Engineering and System Safety.*vol. 183, pp. 240–251, 2019.

[38] M. Hou, D. Pi, and B. Li, "Similarity-based deep learning approach for remaining useful life prediction," *Measurement*, vol. 159, Article ID 107788, 2020.

[39] W. Zhang, G. Peng, C. Li, Y. Chen, and Z. Zhang, "A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals," *Sensors (Switzerland)*, vol. 17, no. 2, 2017.

[40] A. Saxena, M. Ieee, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *Proceedings of the 2008 International Conference on Prognostics and Health Management*, pp. 1–9, IEEE, Denver, CO, USA, October 2008.

[41] A. C. Ian Goodfellow, Y. Bengio, and A. Courville, *Deep Learning Book*, MIT Press, Cambridge MA USA, 2015.

[42] A. Tato and R. Nkambou, "Improving adam optimizer," in *Proceedings of the ICLR 2018 Workshop Submission*, Vancouver, Canada, 2018.

[43] H. Li, W. Zhao, Y. Zhang, and E. Zio, "Remaining useful life prediction using multi-scale deep convolutional neural network," *Applied Soft Computing*, vol. 89, Article ID 106113, 2020.

[44] T. Xia, Y. Song, Y. Zheng, E. Pan, and L. Xi, "An ensemble framework based on convolutional bi-directional LSTM with multiple time windows for remaining useful life estimation," *Computers in Industry*, vol. 115, Article ID 103182, 2020.

[45] F. Pedregosa, A. Gramfort, V. Michel et al., "Scikit-learn: machine learning in Python," *Journal of Machine Learning and Research*, vol. 12, 2011.

[46] T. Berghout, L.-H. Mouss, O. Kadri, L. Saïdi, and M. Benbouzid, "Aircraft engines remaining useful life prediction with an adaptive denoising online sequential extreme learning machine," *Engineering Applications of Artificial Intelligence*, vol. 96, Article ID 103936, 2020.